Commentary on the Paper: Challenges and Methods in Testing the Remote Agent Planner

G. Michael Tong
Code 583
NASA Goddard Space Flight Center
Greenbelt, MD 20771
Gordon.M.Tong@gsfc.nasa.gov

Introduction

The 2nd NASA Workshop on Planning & Scheduling for Space asked reviewers to provide a written commentary on submitted papers. This commentary discusses the following paper: Challenges and Methods in Testing the Remote Agent Planner (the "paper"), which described the approach used to test the software called the Remote Agent planner (the "planner"). The Remote Agent software was used to control the Deep Space 1 (DS1) spacecraft for two days. The Remote Agent planner is a step towards spacecraft autonomy where human intervention is not required.

General Observations

Here are several general observations. First, planning and scheduling systems are difficult to test. Second, flight software operates in a more restricted environment than the ground software. Third, simpler algorithms can be added to the Remote Agent planner for use on specific missions to facilitate testing.

Testing Planning and Scheduling Systems

Typically, planning and scheduling systems receive requests and planning data as inputs and generate a schedule of activities as an output. In evaluating the output, there is no single correct answer or result. Two or more schedules that have different scheduled activities and different activity start times can be deemed correct by the user. In contrast, for a command and telemetry system, there is only one correct bit pattern for a particular sequence of commands, and only one correct result in converting a raw 8-bit telemetry point to engineering units.

The person who tests the planning and scheduling system (the "tester") has the daunting task of evaluating the output data without knowing what the correct answer should be. The software programmer may be able to calculate the correct answer manually by examining the code and analyzing the system requirements; however, the tester does not do this except for simple cases. Even for the software programmer, the manual calculation of the correct answer becomes impractical for a large number of activities on the output schedule.

The Remote Agent planner uses an automated tool to evaluate the correctness of the output schedule. The tool uses an assertions database and first order predicate logic (FOPL). More information about the tool appears in other papers by the authors. However, the planner can generate output that appears to be correct and passes the automated tool checking, yet the planner still contains subtle bugs.

Flight Software Environment

The flight software environment has limitations. Typically, the flight hardware is ten or more years behind the ground hardware in system performance. The flight software and hardware is an embedded system that has a limited amount of main memory and disk storage. Most flight software testing occurs on a special testbed rather than on the actual spacecraft.

Intermittent problems can occur during flight system testing. Repeatable problems are much easier to find and debug than intermittent problems, which are difficult to reproduce. Embedded flight software may not capture good debugging information when an intermittent problem occurs. Software programmers may be unable to solve problems that occur only one time or that occur infrequently because of an interrupt timing condition. The "Test Harness" may not be in place when an intermittent problem occurs. Of course, these mysterious problems are worrisome.

The traditional approach to commanding the spacecraft involves the use of command loads. The science user generates a science plan that specifies a list of targets for the instrument on the spacecraft to observe. A schedule is produced that takes into account the time to take exposures of the target and the time to slew (turning) from one target to another target. A sequence of commands, called a command load, is generated from the schedule and uplinked to the spacecraft. The onboard computer executes the command load, which is the plan or schedule for the spacecraft. The Remote Agent performs these planning and scheduling functions onboard that are traditionally done by ground systems. It represents the trend of the migration of ground functions to the flight software environment.

Simpler Algorithms

The paper described the dilemma of trying to select test cases to cover the "paths through the search space." One possibility would be to simplify the software under test rather than trying to test complicated software. Simpler algorithms could be added to the Remote Agent planner. For example, fixed activity timelines similar to the Space Shuttle Crew Activity Plan can be used to make the testing problem manageable. This approach is suitable only for scheduling applications that employ pre-stored scenarios.

Mission-specific algorithms could be developed. Those algorithms would be designed to facilitate the testing. Perhaps inheritance could be employed to make this approach more general.

Another possibility is to formalize the simulation approach. NASA typically employs simulations to test complicated systems. The formalization would be an extension of the methods described in the said paper, such as "Multiple Variation Test Cases" and test automation tools.

Specific Comments

Specific comments include (1) the description of "tokens" is not clear, (2) the relationship between the use of formal coverage metrics and the testing of heuristics is not clear, and (3) the testing only describes one portion of the flight software.

The paper defines tokens as follows: "The fundamental execution units in the plan are tokens (activities). Tokens also track spacecraft states and resources." However, the names of the tokens in Tables 1 and 2 look like attributes/properties or states, not objects/activities. The initial state tokens in Table 2 do not appear to be compatible with a class member initialization list approach. The technical approach on the use of tokens may be innovative, but it needs more explanation.

The paper advocates the need for formal coverage metrics that measure coverage for a planner domain model. However, the use of heuristics contributes to the testing difficulties. Testing each heuristic individually is straightforward. On the other hand, the interactions among the executions of various heuristics produce a complicated behavior that may result in the generation of an incorrect plan. Sometimes, a heuristic-based algorithm may fail to find a solution. In other words, if the goal is too difficult, there may be no solution for a given situation. It is not clear how formal coverage metrics can help in these cases.

Flight software testing is already expensive. While the paper addresses only the testing of the planner portion of the Remote Agent software, the testing of the other flight software components is also challenging. For example, the paper mentions other functions such as generating a list of targets, calculating maneuvers, performing maneuvers, calculating slews (turning) from one target to another target, performing slews, taking images, and fault detection and recovery. The cost of testing these flight software functions is a factor.

Conclusion

The paper provides an account of the issues in testing a complicated planning system, the approaches and techniques that were used, the technical and conceptual problems that occurred, and possible improvements and research areas. The method for test case selection was discussed, as well as the automation of some parts of the testing process.

Testing an artificially intelligent agent requires grappling with the solution space. The testing techniques described in this paper may be applied to other artificial intelligence applications.